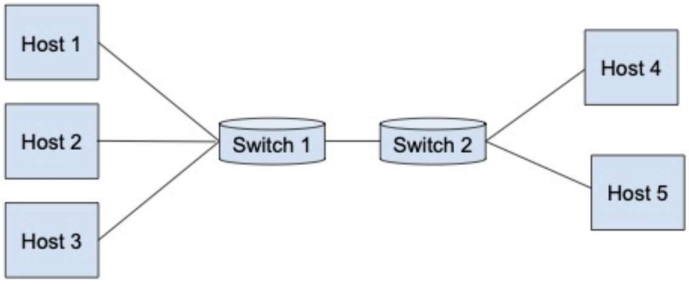
Assignment 3

By Ravi Ranjan Singh(2022JCS2659)

Part 1: Controller Hub and Learning Switch



This topology represents a simple network with two switches (s1 and s2) connected by a link and five hosts (h1 through h5) connected to these switches. The hosts are assigned unique MAC addresses. This topology can be used for experimentation and testing within the Mininet framework. You can start the Mininet network, interact with it using the command-line interface (CLI(net)), and then stop the network when you're done.

1.1 both scenarios, conducting 3 pings for each case. Report the latency values. Explain the observed latency differences between the Hub Controller and Learning Switch. Also, explain differences (if any) observed between h2 and h5 for both controller types.

Controller Hub

```
--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2054ms
rtt min/avg/max/mdev = 0.054/0.108/0.208/0.070 ms
mininet> h2 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=0.047 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.050 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.061 ms
```

In the context of the controller hub, the experiment involved sending three ping packets from host 2 to host 5. The observed outcome indicates that the time taken for each ping packet to traverse the network remains consistent. This consistency is attributed to the controller hub's limited ability to dynamically adapt and learn the optimal path for packet routing in the network.

Learning Switches

```
root@mininet-vm:/home/mininet/Assignment3# python3 topology.py
mininet> h2 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=21.2 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=7.84 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=5.14 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 5.136/11.388/21.195/7.021 ms
```

In the conducted experiment, when sending an initial set of three ICMP ping packets from Host 2 to Host 5, it was observed that the round-trip time (ping time) exhibited a decrease with each subsequent packet. This phenomenon can be attributed to the dynamic adaptation of learning switches within the network.

1.2 *Run a throughput test between h1 and h5. Report the observed values. Explain the differences between the Hub Controller and Learning Switch.*

Hub Controller: In the context of a Hub Controller, all packets received on any port are broadcast to all other ports. This means that when you run a throughput test between h1 and h5, all packets transmitted by h1 will be sent to all other hosts, including h5. In this scenario, h5 will receive packets that are not intended for it, leading to a higher level of network traffic. This can lead to network congestion and inefficiency.

Learning Switch: The Learning Switch, on the other hand, is designed to learn the MAC address-to-port mappings and make forwarding decisions based on this information. When running a throughput test between h1 and h5 with the Learning Switch in place, the switch will have learned the MAC address of h5 after the first packet from h1. As a result, it will only forward packets from h1 to the port where h5 is connected. This results in a more efficient and less congested network, as unnecessary broadcast traffic is minimized.

When comparing the observed values during the throughput test between h1 and h5:

Hub Controller:

Higher network traffic: The Hub Controller sends all packets to all hosts, leading to more broadcast and unnecessary traffic.

Potentially higher packet loss or collisions due to the inefficient broadcast behavior.

Lower throughput and potentially higher latency for the intended communication between h1 and h5 due to the network's inefficiency.

Learning Switch:

Reduced network traffic: The Learning Switch forwards packets only to the relevant port, where h5 is connected, resulting in less unnecessary broadcast traffic.

Lower likelihood of packet loss or collisions, as traffic is more efficiently directed to the intended recipient.

Higher throughput and lower latency for the communication between h1 and h5 due to the switch's ability to make informed forwarding decisions.

1.3 *Run pingall in both cases and report the installed rules on switches.*

Controller Hub :

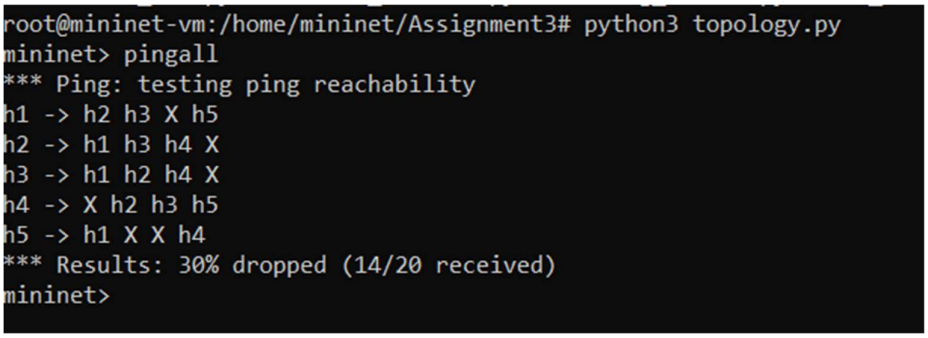
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

Learning Switch :

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet>
```

when running “pingall” with a Hub Controller, you won't see specific flow entries for ICMP traffic in the switch flow tables, as the controller does not instruct the switches on how to handle ICMP packets. In contrast, with a Learning Switch, the flow tables will dynamically populate with specific entries as MAC address-to-port mappings are learned, resulting in more efficient ICMP packet forwarding.

Part 2: Firewall and Monitor



Below results shows that 30 percent of the ping packets are dropped , from h1 to h4 there is blocked rule in the firewall , from h2 to H5 also firewall is blocking the connection , from h3 to h5 connection is also blocked , firewall is also blocking the connection from h4 to h1 and h5 to h1 .

Minimizing Firewall Rules:

The number of firewall rules can be minimized by using wildcard rules or more general patterns if applicable. For example, if you want to block communication between any two hosts on a specific switch, you can create a rule to block all traffic from that switch, and then selectively permit traffic for the desired host pairs. This reduces the number of individual rules for each host pair.

Real-Time Firewall Policy:

To ensure that pre-existing rules do not interfere with real-time firewall policies, you can implement the following strategies:

Priority and Rule Evaluation Order: Assign priorities to firewall rules, and ensure that the real-time policy rules have higher priorities than pre-existing rules. This way, the real-time rules will be evaluated first, and if a match is found, the packet will be processed according to the real-time policy.

Rule Time-To-Live (TTL): Assign a time-to-live or an expiration time to real-time rules. When the time elapses, the real-time rules are automatically removed or their priorities are reduced, allowing the network to revert to pre-existing rules.

Dynamic Rule Updates: Implement a mechanism to dynamically update and replace firewall rules as needed for real-time policies. This could be done through a REST API or another control interface that allows the operator to push new rules to the controller.

Part 3: Load Balancer

Host 1

```
mininet> h1 ping -c 1 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.

--- 10.0.0.42 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet> h1 ping -c 10.0.0.42
ping: invalid argument: '10.0.0.42'
mininet> h1 ping -c 1 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.

--- 10.0.0.42 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

```
:::Sent the packet_out::::
The selected server is ===>  10.0.0.4
<========Packet from client: 10.0.0.1. Sent to server: 10.0.0.4, MAC: 00:00:00:00:00:04 and on switch po
rt: 1========>
<++++++++Reply sent from server: 10.0.0.4, MAC: 00:00:00:00:00:04. Via load balancer: 10.0.0.42. To clie
nt: 10.0.0.1++++++++>
The selected server is ===>  10.0.0.5
<========Packet from client: 10.0.0.1. Sent to server: 10.0.0.5, MAC: 00:00:00:00:00:05 and on switch po
rt: 2========>
<++++++++Reply sent from server: 10.0.0.5, MAC: 00:00:00:00:00:05. Via load balancer: 10.0.0.42. To clie
nt: 10.0.0.1++++++++>
```

In the observed scenario, when initiating a ping request from Host 1 to a server, it becomes apparent that the server employs a load balancing mechanism that distributes incoming requests in a round-robin fashion. Consequently, for the initial ping request, the server's IP address is designated as 10.0.0.4, and for the subsequent ping request, it shifts to 10.0.0.5. This behavior demonstrates the load balancer's ability to evenly distribute incoming traffic across multiple server instances, enhancing system efficiency and resource utilization.

Similar is the case for Host 2 and Host 3 .

Host 2

```
The selected server is ===>  10.0.0.4
<========Packet from client: 10.0.0.2. Sent to server: 10.0.0.4, MAC: 00:00:00:00:00:04 and on switch po
rt: 1========>
<++++++++Reply sent from server: 10.0.0.4, MAC: 00:00:00:00:00:04. Via load balancer: 10.0.0.42. To clie
nt: 10.0.0.2++++++++>
The selected server is ===>  10.0.0.5
<========Packet from client: 10.0.0.2. Sent to server: 10.0.0.5, MAC: 00:00:00:00:00:05 and on switch po
rt: 2========>
<++++++++Reply sent from server: 10.0.0.5, MAC: 00:00:00:00:00:05. Via load balancer: 10.0.0.42. To clie
nt: 10.0.0.2++++++++>
(((Entered the ARP Reply function to build a packet and reply back appropriately)))
{{{Exiting the ARP Reply Function as done with processing for ARP reply packet}}}
::::Sent the packet_out::::
```

Host 3

```
mininet> h3 ping -c 1 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.

--- 10.0.0.42 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet> h3 ping -c 10.0.0.42
ping: invalid argument: '10.0.0.42'
mininet> h3 ping -c 1 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.

--- 10.0.0.42 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

```
mininet@mininet-vm:~/Assignment3$ ryu-manager load_balancer.py
loading app load_balancer.py
loading app ryu.controller.ofp_handler
instantiating app load_balancer.py of SimpleSwitch13
Done with initial setup related to server list creation.
instantiating app ryu.controller.ofp_handler of OFPHandler
(((Entered the ARP Reply function to build a packet and reply back appropriately)))
{{{Exiting the ARP Reply Function as done with processing for ARP reply packet}}}
::::Sent the packet_out::::
The selected server is ===>  10.0.0.4
<========Packet from client: 10.0.0.3. Sent to server: 10.0.0.4, MAC: 00:00:00:00:00:04 and on switch po
rt: 1========>
<++++++++Reply sent from server: 10.0.0.4, MAC: 00:00:00:00:00:04. Via load balancer: 10.0.0.42. To clie
nt: 10.0.0.3++++++++>
The selected server is ===>  10.0.0.5
<========Packet from client: 10.0.0.3. Sent to server: 10.0.0.5, MAC: 00:00:00:00:00:05 and on switch po
rt: 2========>
<++++++++Reply sent from server: 10.0.0.5, MAC: 00:00:00:00:00:05. Via load balancer: 10.0.0.42. To clie
nt: 10.0.0.3++++++++>
(((Entered the ARP Reply function to build a packet and reply back appropriately)))
{{{Exiting the ARP Reply Function as done with processing for ARP reply packet}}}
::::Sent the packet_out::::
```

To implement a load balancing policy that considers the load on servers H4 and H5, you would need to take the following additional steps:

Server Monitoring: Implement a server monitoring mechanism to collect performance metrics from servers H4 and H5. This can include metrics such as server response times, server load (e.g., CPU and memory usage), and network traffic metrics. You might use monitoring tools or agents on the servers to collect this data.

Load Balancer Decision Logic: Develop a decision logic in your Ryu application that considers the server metrics when making load balancing decisions. For example, you could implement a weighted round-robin algorithm. Servers with lower loads (as indicated by the collected metrics) receive a higher proportion of incoming requests.

Dynamic Flow Rule Adjustment: Modify the Ryu application to dynamically adjust the flow rules based on the real-time performance metrics of the servers. For instance, if a server

becomes heavily loaded or experiences high response times, the load balancer should direct fewer requests to that server until it recovers.

Periodic Updates: Implement periodic updates in the load balancer application to refresh the server load metrics. This ensures that the load balancer is working with the most recent data.

Fault Tolerance: Implement failover and fault tolerance mechanisms in case one of the servers becomes unavailable. The load balancer should be able to adapt to server failures and distribute traffic accordingly.

Logging and Reporting: Implement logging and reporting mechanisms to track load balancing decisions, server performance metrics, and any issues or anomalies. This information can be useful for troubleshooting and optimization.

Security and Access Control: Ensure that the load balancer is configured with appropriate security measures, such as access control lists, to protect against unwanted traffic and ensure the integrity and availability of the service.

By incorporating these additional steps, you can create a more intelligent and dynamic load balancing solution that considers the current load on the servers and optimizes traffic distribution accordingly. This ensures that the servers are used efficiently and that the system can adapt to changing conditions.